

UNIVERSIDADE DO MINHO

MESTRADO EM ENGENHARIA INFORMÁTICA
APRENDIZAGEM PROFUNDA

Geração de música com modelos generativos



Henrique Parola
pg50415



Ana Henriques
pg50196

12 de junho de 2023



Conteúdo

1	Introdução	2
2	Modelo Generativo: LSTM	2
2.1	Importação dos dados	3
2.2	Exploração dos dados	3
2.3	Tratamento dos dados	4
2.4	Construção do modelo neuronal	5
2.5	Análise das métricas de avaliação	6
2.6	Geração da Música	7
2.7	<i>Pipeline</i> de execução	7
3	Interface Gráfica	8
4	Conclusão e Trabalho Futuro	9
A	Gráficos Obtidos	10
B	Excertos de Código	12
B.1	Extração da lista de notas para formar um <i>corpus</i>	12
B.2	Mapeamento das notas únicas com os seus repetitivos índices	12
B.3	Codificação do <i>corpus</i> como <i>labels</i> e <i>targets</i>	13
B.4	Geração de uma melodia a partir de notas aleatórias	14



1 Introdução

A crescente disponibilidade e acessibilidade da tecnologia têm impulsionado o interesse em abordagens inovadoras para diversas áreas, como a criação musical. No entanto, de acordo com Huang et al. [1], o custo da produção musical ainda é bastante elevado. Como proposta para solucionar esta adversidade, a composição de música por meio da inteligência artificial tem se revelado uma alternativa promissora, proporcionando maior diversidade e rapidez no processo de criação.

À medida que exploramos as implicações fascinantes da geração de música, é importante reconhecer que, tradicionalmente, esta prática é vista como uma habilidade exclusiva de profissionais na área. Porém, segundo Liu et al. [2], mesmo aqueles sem formação musical têm a capacidade de criar melodias simples, baseadas nas suas próprias emoções e sentimentos. Através do uso de técnicas de *Machine Learning*, podemos implementar modelos generativos capazes de aprender e reconhecer padrões musicais. Ao fornecer novos caminhos criativos, esses modelos podem motivar qualquer pessoa, independentemente da sua formação musical, a participar ativamente na produção de música.

O objetivo deste projeto é, portanto, gerar música, mais especificamente faixas de piano, utilizando um modelo generativo LSTM. Assim sendo, a nossa solução passa por implementar uma aplicação *web* para que pessoas comuns possam criar as suas próprias músicas, com base em composições existentes nesta indústria. Primeiramente, serão mencionados os tratamentos de dados efetuados, seguidos pela construção e análise do modelo. Deste modo, o presente relatório fornece um *roadmap* de todo o trabalho desenvolvido, desde as escolhas até às decisões finais feitas sobre o modelo.

2 Modelo Generativo: LSTM

No decorrer do projeto, o grupo procurou uma solução já existente para a geração de música através de modelos generativos. Como tal, explorámos a plataforma *Kaggle*, conhecida pela sua ampla variedade de recursos e modelos específicos nesta área. Após uma pesquisa minuciosa, encontrámos um modelo¹ desenvolvido com base numa rede neural recorrente (RNN), mais especificamente uma **LSTM**², que utiliza a biblioteca *music21* do *Python* para o processamento de dados musicais.

Com o objetivo de aprimorar a proposta encontrada, o modelo sofreu algumas modificações, nomeadamente a incorporação de outras características musicais para além do ***pitch***, como a **duração**, o **passo** e a **oitava**. Chacón et al. [3] ressaltam que a exploração de vários parâmetros musicais é essencial ao criar música, uma vez que tornam o conteúdo mais expressivo e dinâmico. Para não entendedores de música, enquanto que o passo é o intervalo entre uma nota e a nota imediatamente subjacente, a oitava é o intervalo entre uma nota e outra com metade ou o dobro da sua frequência. Para acomodar essas adições, foi necessário acrescentar novas camadas à arquitetura. Essas modificações proporcionaram uma base sólida para a implementação do modelo generativo, tornando-o capaz de produzir composições diversificadas. O ***notebook final***, que incorpora todas as *features* referidas, denomina-se por *Music-Learning-Octave.ipynb*.

¹<https://www.kaggle.com/code/karnikakapoor/music-generation-lstm/notebook>

²*Long Short-Term Memory*



2.1 Importação dos dados

O projeto original¹ tem acesso a um conjunto de ficheiros MIDI, correspondentes a músicas clássicas de renomeados artistas como Beethoven, Mozart e Chopin. Assim sendo, numa fase inicial, o grupo usou como *dataset* cerca de 30 composições do Beethoven.

Como ponto de partida, criámos uma lista com todas as composições selecionadas. Em seguida, extraímos tanto acordes como notas do *dataset*, transformando-os num *corpus*, como se pode ver no excerto B.1. É importante destacar a diferença entre notas e acordes. Uma **nota musical** é um som único e distinto produzido por um instrumento, como o som de uma única tecla de piano. Por outro lado, um **acorde** é uma combinação de três ou mais notas tocadas simultaneamente, criando uma sonoridade mais rica e complexa.

A ideia é que a lista criada contenha uma combinação de acordes e notas individuais. O nosso objetivo é, então, extrair esses elementos especificamente como notas individuais, a fim de obter uma série de notas que formem a composição musical. Isso permitirá uma análise mais detalhada para gerar música a partir do modelo que estamos a desenvolver.

Para enriquecer ainda mais o nosso conjunto de dados, decidimos criar um *dataset* que incluísse músicas de outros artistas e géneros musicais, além dos clássicos. Contudo, ao enfrentar o desafio de encontrar arquivos MIDI *online* que correspondessem às nossas necessidades, tomámos a decisão de criar os nossos próprios arquivos MIDI.

Primeiramente, procurámos versões de músicas em piano no *YouTube*, prosseguindo para a sua conversão em formato .mp3. De modo a garantir a sua qualidade, recorremos à ferramenta *Audacity* para filtrar as músicas, removendo ruídos indesejados e ajustando o volume. O passo seguinte é converter os ficheiros de áudio .mp3 para o formato MIDI, através do conversor *open-source* do *Spotify*, o *Basic Pitch*. Esta plataforma é capaz de analisar o áudio e extrair as informações mais relevantes, como *pitches*, durações e outros elementos cruciais para a criação dos arquivos MIDI. Um ponto interessante sobre este *software* é que ele concretiza esta criação através de algoritmos de inteligência artificial.

2.2 Exploração dos dados

As notas musicais, fundamentais para a expressão melódica, podem ser compreendidas como ondas sonoras que se propagam pelo espaço. No contexto musical, essas ondas adquirem características particulares, como a frequência e o comprimento de onda, que são padronizadas e denominadas como notas. No nosso *corpus*, reside a identificação dos nomes dessas notas, o que nos permite explorar a estrutura musical de forma mais precisa. Ao carregarmos os dados com a biblioteca *music21*, conseguimos informações valiosas a respeito de cada nota. Além das próprias características sonoras, temos também dados sobre a duração das notas e outras informações relevantes para a análise musical.

Tal como citado previamente, o grupo usou como *dataset* cerca de 30 composições do Beethoven. Esse conjunto de dados é composto por mais de 80000 notas, entre as quais existem 349 *pitches* únicos, 33 durações únicas, 7 passos únicos e 7 oitavas únicas. Isto revela alguma repetição de notas ao longo das composições, o que é uma característica comum na música. De facto, certas notas ou sequências melódicas são recorrentes para criar harmonia e estrutura musical. Ao explorá-las, podemos identificar padrões, que são essenciais para a construção de uma peça de música coesa.



Em contrapartida, a presença de notas raras no *corpus* pode levantar problemas e erros durante o processo de composição. Como forma de mitigar esses impactos negativos, é necessário examinar a frequência das notas no *corpus*. Perante isto, optámos por excluir as notas menos frequentes, i.e., as que aparecem menos de 100 vezes, a fim de garantir uma geração musical mais consistente. Ao analisar o *dataset*, verifica-se que há um total de 81272 sequências de notas. A frequência média de uma nota é aproximadamente 232,98, atendendo que a unidade mais recorrente apareceu 2094 vezes.

2.3 Tratamento dos dados

Na subsecção anterior, averiguou-se que a existência de *outliers*³ num contexto específico (neste caso, notas raras) pode prejudicar modelos de *Deep Learning*. Visando a solução deste problema, decidimos substituir as notas raras pela mais frequente e, desta maneira, preencher as lacunas nas sequências musicais para obter um resultado mais harmonioso.

O código seguinte representa precisamente esta substituição. Se as notas únicas do *corpus*, guardadas no dicionário `count_num` juntamente com o seu número de aparições, ocorrerem mais de 100 vezes⁴, serão guardadas numa lista para posterior tratamento.

```
frequent_notes = []
for note, count in count_num.items():
    if count >= 100:
        frequent_notes.append(note)
```

De seguida, ao percorrer cada elemento do *corpus*, as notas que tiverem sido previamente consideradas como raras são substituídas pelo elemento da lista `frequent_notes`, cuja contagem é maior do que a de qualquer outra nota.

```
for i in range(len(Corpus)):
    if Corpus[i] in rare_note:
        Corpus[i] = max(frequent_notes, key=lambda x: count_num[x])
```

Para que as informações contidas nas notas possam ser adequadamente compreendidas, estabelecemos um **mapeamento** entre as notas e os seus respectivos índices. Como os sistemas computacionais interpretam os nomes das notas como símbolos, é preciso associar cada nota única a um número específico. Essa relação de correspondência será essencial para a codificação e decodificação desses dados ao interagirmos com a RNN. Assim sendo, por meio do mapeamento B.2, é possível traduzir as propriedades musicais em representações numéricas, comprehensíveis pela RNN. Isto é um passo fundamental para que a rede possa aprender padrões, realizar previsões e gerar sequências coerentes.

Interessados em estudar o potencial do *corpus*, segue-se a sua **codificação** B.3 de maneira a torná-lo mais apropriado ao treino da rede. Como tal, convertermos o *corpus* em sequências de igual comprimento, intentando uma padronização do formato. Cada sequência é composta por características musicais e pelos seus respectivos *targets*. As *labels* passam por um processo de **reshaping** e **normalização**, com o intuito de melhorar a representação e facilitar o treino do modelo. Já os *targets*, por sua vez, são codificados

³Dados que se afastam significativamente do padrão geral do *dataset*

⁴Valor atribuído pela equipa com base no Gráfico 16



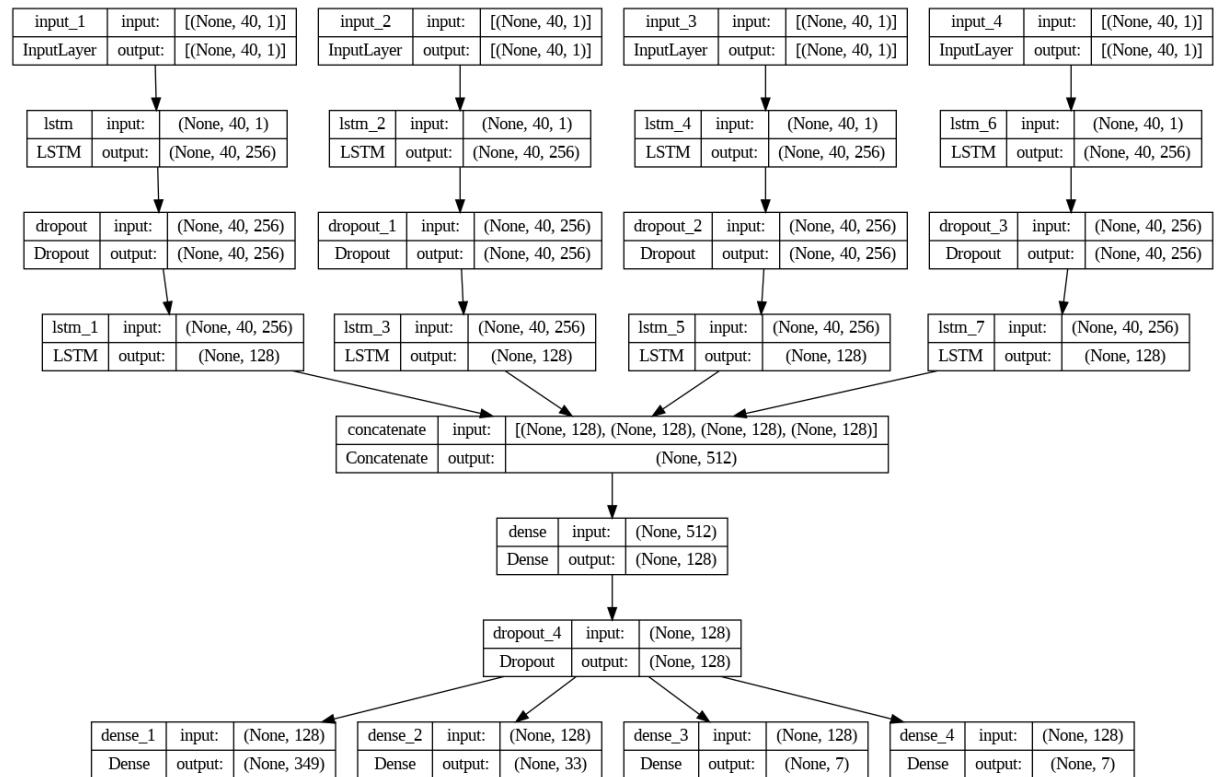
usando a técnica de *one-hot encoding*, que os transformará em vetores binários cujos elementos correspondem a uma classe específica. Isto permite que a RNN aprenda as relações entre as diferentes características musicais.

A criação de música através de uma RNN carece do fornecimento de uma entrada inicial. Para isso, reservamos uma parte dos dados do *corpus* como "sementes" para a geração de números aleatórios, que influenciam diretamente as sequências geradas posteriormente. Isso garante que a divisão de 20% dos dados para teste seja reproduzível. Por outras palavras, ao executar o código várias vezes com a mesma semente, obteremos a mesma divisão dos dados. O valor escolhido arbitrariamente para a semente foi 42.

2.4 Construção do modelo neuronal

A arquitetura da rede neuronal concebida possui, nomeadamente, três níveis de processamento. O primeiro nível sustenta-se em *branches* com duas camadas LSTM, individualizadas para cada uma das 4 *features* das notas (*pitch*, passo, duração e oitava). Como pode ser visto na Figura 1, a camada de *input* inicial do modelo processa elementos tridimensionais (exigido pelas camadas LSTM), com a segunda dimensão com tamanho 40, uma vez que este é o tamanho das sequências usadas para treino.

O segundo nível da rede destina-se à concatenação, em conjunto com uma camada densa de nodos. A necessidade deste nível é poder **relacionar** as 4 diferentes *features*. Sem essa camada, seria como se houvesse 4 modelos separados para cada característica de uma nota musical, o que poderia resultar na não captura dos padrões musicais.



Finalmente, o terceiro e último nível corresponde às camadas de *output* para cada *feature*. Como pode ser observado na figura acima, o tamanho destas camadas depende da quantidade de elementos únicos dos seus respectivos *corpus* (devido à realização de *one-hot encoding* sobre os *targets*). Pode-se, ainda, dizer que os dados previstos pelo modelo vão indicar a probabilidade de cada *feature* assumir um determinado valor dentro dos presentes no seu *corpus*. Por este motivo, é utilizada a função *softmax* como função de ativação para a última camada. Adicionalmente, o modelo foi configurado com o otimizador *Adamax* com um *learning rate* de 0.01.

2.5 Análise das métricas de avaliação

Para a análise das métricas de avaliação de *loss* e *accuracy*, o grupo guiou-se exclusivamente pelos gráficos gerados durante o treino do modelo para o artista Beethoven. O gráfico apresentado na Figura 2 ilustra apenas o *pitch* de cada nota no *corpus* musical, conforme demonstrado no *notebook* original. Por outro lado, o gráfico mostrado na Figura 3 avalia todas as características adicionadas ao modelo.

Como se pode analisar, na presença de outras características musicais, ambas as métricas de avaliação do *pitch* evoluem mais gradualmente, em comparação ao modelo que considera apenas essa *feature* isoladamente. No entanto, também se ressalta que o *pitch* não é tão preciso quanto a duração e o passo das notas, por exemplo. Enquanto a *accuracy* do *pitch* alcança cerca de 75%, a do passo consegue quase atingir os 100%. A partir de uma determinada época, estes números tendem a estagnar, revelando pouca variação.

É fundamental sublinhar que estes valores são específicos para o Beethoven, cujo conjunto de dados é extenso o suficiente para produzir resultados satisfatórios. Porém, ao considerar outros artistas, é provável que estes valores variem substancialmente.

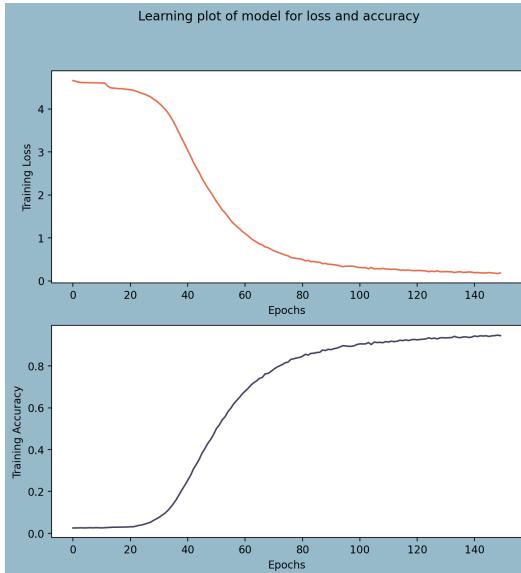


Figura 2: Métricas com apenas a *feature* *pitch* (projeto original)

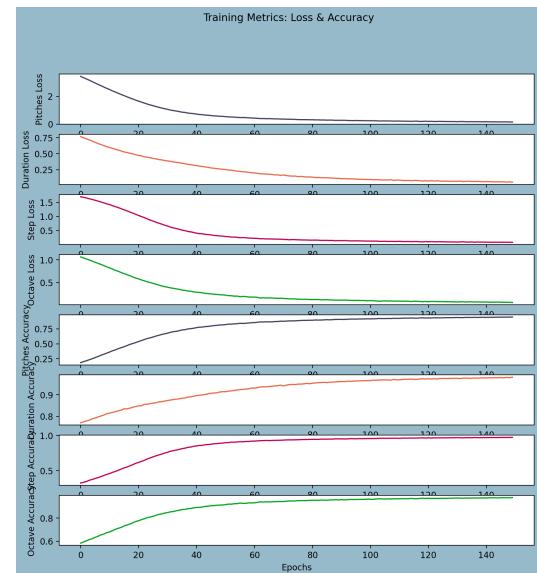


Figura 3: Métricas com todas as *features* adicionadas



2.6 Geração da Música

A geração da música é realizada a partir da seleção inicial de uma sequência de notas, **não utilizada no treino da rede** como *seed* para a música gerada. Isto é, recorrendo a esta sequência de 39 notas, é feita a previsão da nota 40 e o seu respetivo *append* na sequência sorteada. Através dos últimos 39 elementos da sequência resultante (já com a nota prevista e sem a primeira nota) é, então, produzida iterativamente uma nova nota musical, até termos cerca de 50 segundos de música.

Um detalhe pertinente é que o modelo não gera diretamente as notas musicais. Como foi explicado anteriormente, as notas musicais foram mapeadas em valores numéricos compreendidos pela rede neuronal. Porém, para utilizar a biblioteca *music21* na construção das novas músicas, seria preciso obtermos os valores concretos dos atributos das notas. Portanto, a partir dos valores previstos pelo modelo, são obtidos os respectivos *pitches*, durações, passos e oitavas, usando os mapeamentos das notas musicais e os seus índices descritos na subsecção 2.3. O código da função *Melody_Generator*, responsável por este processo de criação, está disponível em [B.4](#).

Quanto aos resultados, podemos concluir que a complexidade e qualidade da música gerada aumentou conforme a consideração de mais *features* das notas musicais. Estes resultados estão disponíveis no [repositório da aplicação](#).

2.7 Pipeline de execução

O processo de desenvolvimento do modelo, desde a importação dos dados até a geração das faixas de piano, é sumariado no Esquema 4. Como foi explicado, a criação das sequências de treino, a partir dos ficheiros MIDI, e a conversão das sequências geradas pelo modelo são etapas **simétricas** uma da outra. Ambas as fases utilizam a biblioteca *music21*.

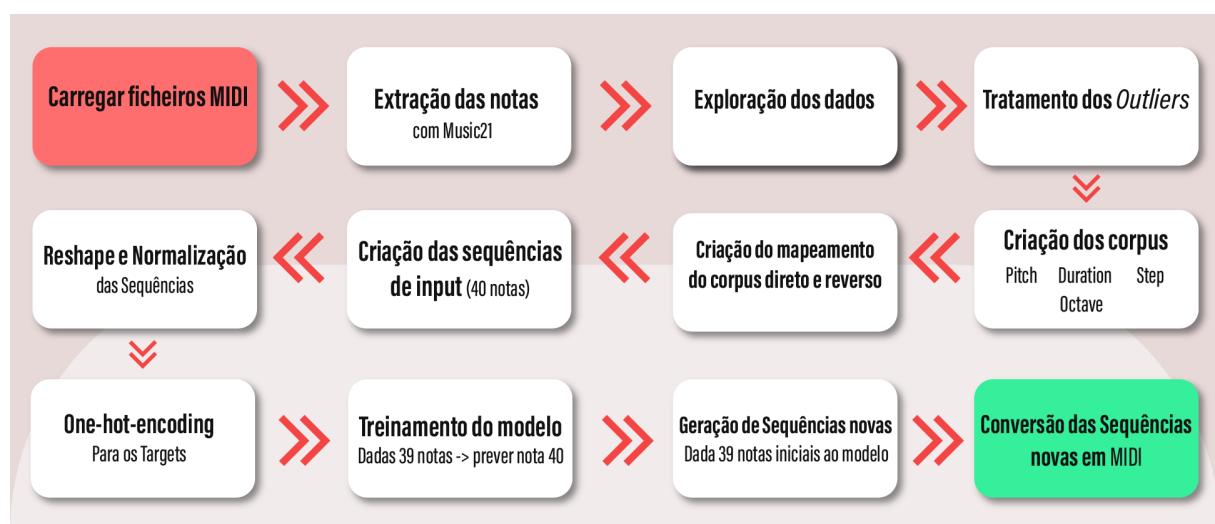


Figura 4: *Pipeline* de Execução



3 Interface Gráfica

No âmbito da geração de música com modelos generativos, o grupo ambicionou não apenas desenvolver algoritmos de *Deep Learning*, como também proporcionar uma experiência envolvente para o usuário. O resultado disso foi a implementação de uma interface gráfica, que amplifica todas as funcionalidades disponíveis. Este ambiente interativo foi conseguido com o uso da biblioteca *Flask* do *Python*, que oferece uma estrutura flexível para o desenvolvimento de aplicações *web*, e da linguagem *HTML*.

O modo de funcionamento é bastante simples e intuitivo. A página inicial da aplicação *web*, ilustrada na Figura 5, apresenta as 4 alternativas de artistas disponíveis: Halsey, Beethoven, Madonna e Chopin. Adicionalmente, no canto inferior direito, dispomos de um botão que concede acesso a uma secção informativa – Figura 6, na situação do usuário estar interessado em saber mais sobre a aplicação em questão.

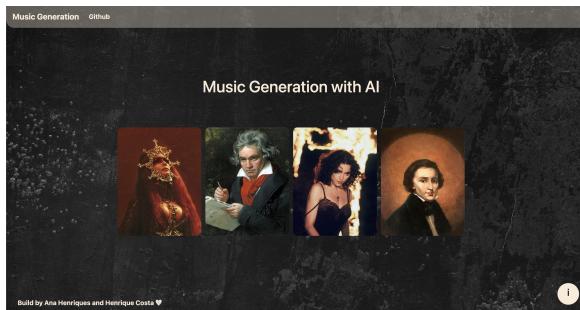


Figura 5: Página inicial

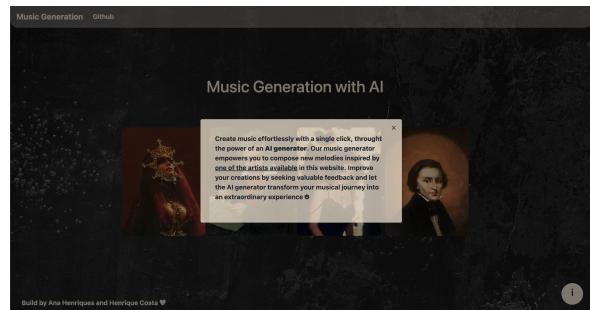


Figura 6: Secção de informações

A ideia subjacente é que, ao selecionar um dos artistas, o aplicativo exibe um indicador de carregamento, como exemplificado na Figura 7, a fim de simular o processo de criação musical, já previamente realizado. Assim que finalizado, o usuário será redirecionado para a página composta pela música gerada – Figura 8. Além de identificar o artista selecionado, essa página também disponibiliza uma barra de reprodução de áudio, permitindo ao usuário reproduzir, pausar, avançar e retroceder na música.

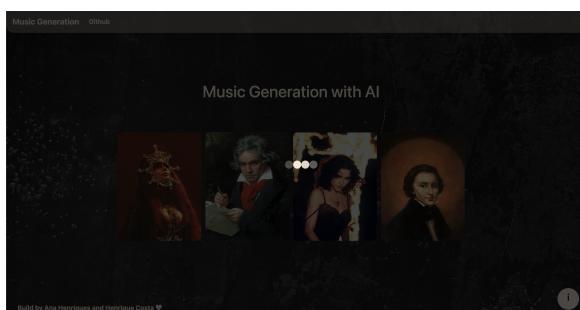


Figura 7: Gerando música do artista

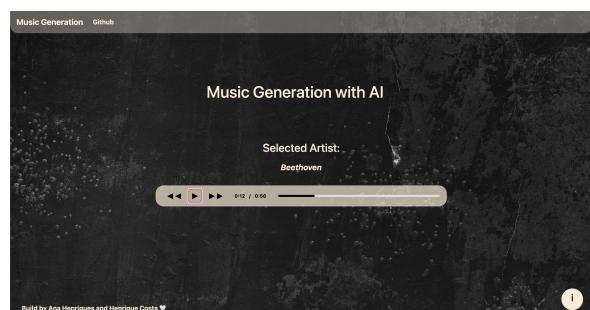


Figura 8: Página com a música gerada

A presente aplicação *web* está disponível para acesso público, basta visitar este [link](#).



4 Conclusão e Trabalho Futuro

Uma primeira análise crítica a ser feita é a dependência da qualidade das músicas geradas com a qualidade dos ficheiros MIDI usados como *input* para o modelo. Na realidade, ter mais ficheiros MIDI não significa necessariamente um melhor modelo. Isto denotou-se graças à discrepância na qualidade das composições produzidas entre os modelos do Beethoven e do Chopin e os modelos da Halsey e da Madonna. No segundo caso, os ficheiros MIDI foram extraídos manualmente pela equipa de trabalho e, por isso, não possuía uma precisão tão boa no que concerne à estrutura musical, comparativamente com os ficheiros do Beethoven e do Chopin. Esta diferença de qualidade refletiu-se tanto a nível objetivo (métricas de avaliação analisadas), como subjetivo (gosto musical). Os gráficos das métricas de avaliação para os modelos do Chopin, da Halsey e da Madonna podem ser consultados nas Figuras 13, 14 e 15, respetivamente.

No que respeita a interface *web* e modelos gerados, uma sugestão para trabalho futuro é a adição de um mecanismo de *feedback* sobre as músicas geradas. A interação dos utilizadores com a aplicação poderá trazer melhorias no modelo de aprendizagem. Com isto, seria possível etiquetar as composições que os utilizadores julgam ser **boas** ou **máis**. A razão desta adição reside justamente na carência de métricas de avaliação nesta área. Avaliar objetivamente a qualidade de uma música composta ainda é uma zona cinzenta em *Deep Learning*. Por este motivo, reconhecemos que a precisão na previsão das notas apresentada como forma de avaliação de desempenho não é um indicador direto da qualidade dos resultados.

Neste momento, foram concebidos modelos individualizados para cada artista. Uma outra sugestão para trabalho futuro seria a junção de estilos musicais durante o treino do modelo. Com algumas modificações nos dados de entrada e na arquitetura da rede, poderíamos até conceber um modelo capaz de gerar composições com base na descrição dos estilos musicais pretendidos pelo utilizador.

A utilização das *branches* LSTM para cada *feature* musical, seguidas das suas respectivas concatenações, mostrou-se uma boa abordagem de solução arquitetural para o problema. Poder-se-á, no futuro, testar novas configurações da rede, como uma única camada LSTM desde o começo até o fim da rede, com o objetivo de capturar ainda a relação de cada atributo das notas musicais.

A Gráficos Obtidos

Evolução das métricas de avaliação do treino do modelo do Beethoven, com a adição de novas *features*

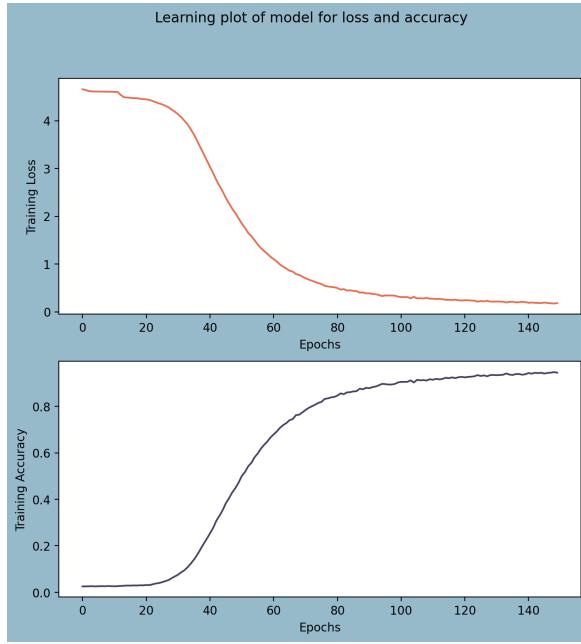


Figura 9: Métricas de avaliação para apenas a *feature pitch*

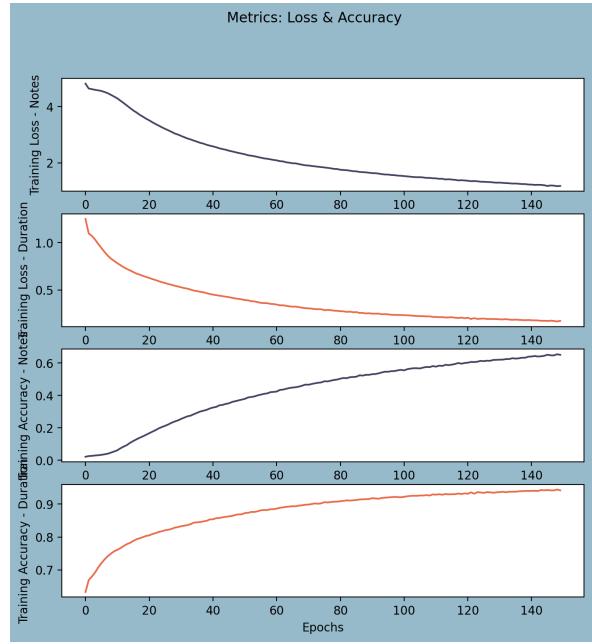


Figura 10: Métricas de avaliação com uma nova *feature*: duração

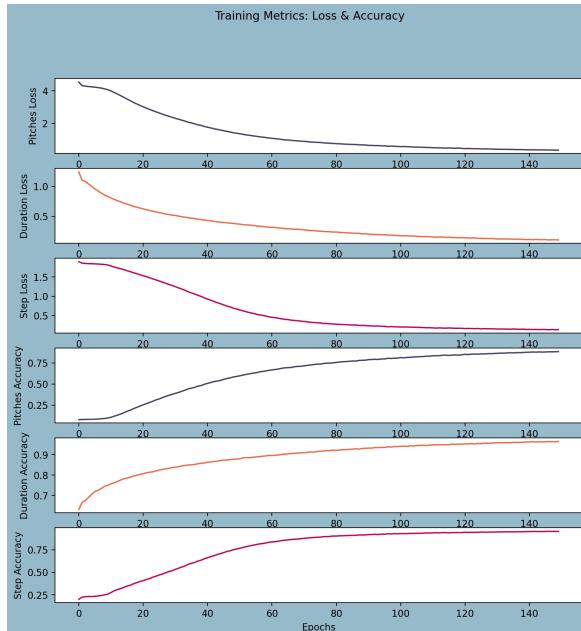


Figura 11: Métricas de avaliação com uma nova *feature*: passo

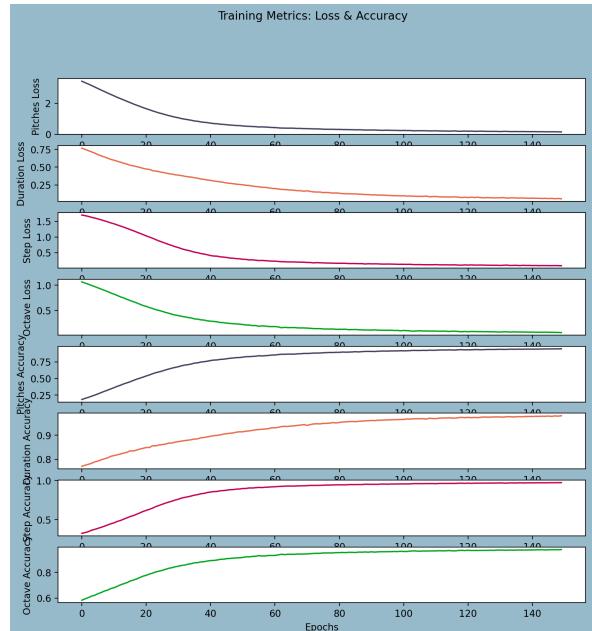


Figura 12: Métricas de avaliação com uma nova *feature*: oitava

Métricas de avaliação do treino dos modelos do Chopin, da Halsey e da Madonna, respectivamente

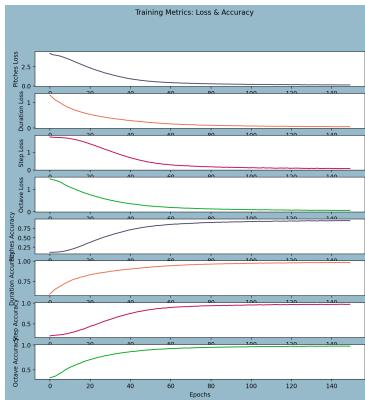


Figura 13: Métricas do treino do modelo da Chopin

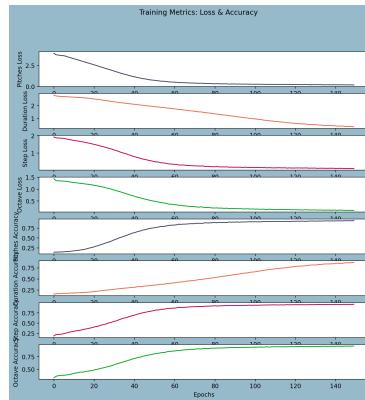


Figura 14: Métricas do treino do modelo da Halsey

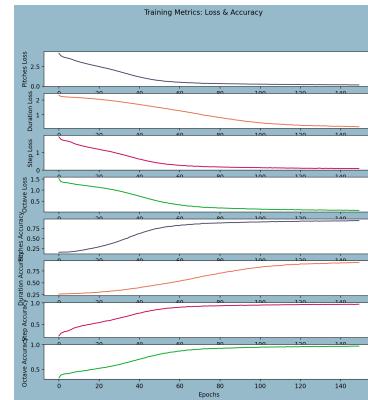


Figura 15: Métricas do treino do modelo da Madonna

Frequência das notas no *corpus*

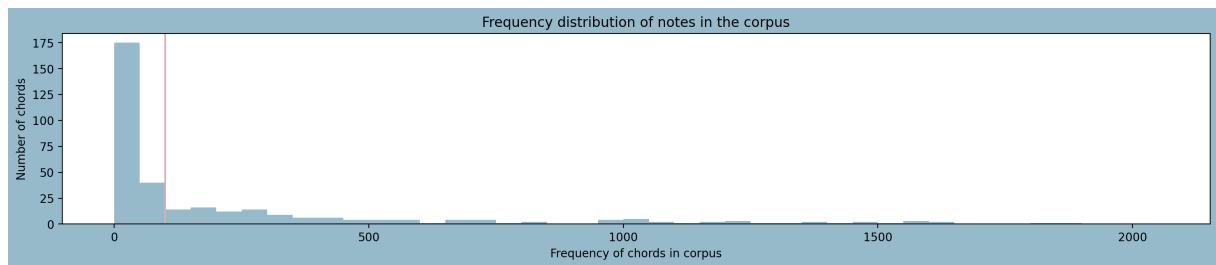


Figura 16: Distribuição da frequência das notas no *corpus*



B Excertos de Código

B.1 Extração da lista de notas para formar um *corpus*

```
def extract_notes(file):
    pitches = []
    durations = []
    steps = []
    octaves = []
    pick = None
    for j in file:
        songs = instrument.partitionByInstrument(j)
        for part in songs.parts:
            pick = part.recurse()
            for element in pick:
                if isinstance(element, note.Note):
                    notes.append(str(element.pitch))
                    durations.append(element.duration.quarterLength)
                    steps.append(str(element.step))
                    octaves.append(str(element.octave))
                elif isinstance(element, chord.Chord):
                    notes.append(".".join(str(n) for n in element.normalOrder))
                    durations.append(element.duration.quarterLength)
                    steps.append(str(element.step))
                    octaves.append(str(element.octave))

    return notes, durations, steps, octaves

Corpus_pitches, Corpus_durations, Corpus_steps, Corpus_octaves
= extract_notes(all_midis)
```

B.2 Mapeamento das notas únicas com os seus repetitivos índices

```
symb_notes = sorted(list(set(Corpus_pitches)))
symb_durations = sorted(list(set(Corpus_durations)))
symb_steps = sorted(list(set(Corpus_steps)))
symb_octaves = sorted(list(set(Corpus_octaves)))

L_pitches = len(Corpus_pitches)
L_durations = len(Corpus_durations)
L_steps = len(Corpus_steps)
L_octaves = len(Corpus_octaves)

L_symb_notes = len(symb_notes)
L_symb_durations = len(symb_durations)
L_symb_steps = len(symb_steps)
L_symb_octaves = len(symb_octaves)
```



```
mapping_notes = dict((c, i) for i, c in enumerate(symb_notes))
mapping_durations = dict((c, i) for i, c in enumerate(symb_durations))
mapping_steps = dict((c, i) for i, c in enumerate(symb_steps))
mapping_octaves = dict((c, i) for i, c in enumerate(symb_octaves))

reverse_mapping_notes = dict((i, c) for i, c in enumerate(symb_notes))
reverse_mapping_durations = dict((i, c) for i, c in enumerate(symb_durations))
reverse_mapping_steps = dict((i, c) for i, c in enumerate(symb_steps))
reverse_mapping_octaves = dict((i, c) for i, c in enumerate(symb_octaves))

B.3 Codificação do corpus como labels e targets

length = 40

note_features = []
duration_features = []
step_features = []
octave_features = []

note_targets = []
duration_targets = []
step_targets = []
octave_targets = []

for i in range(0, L_pitches - length, 1):
    note_feature = Corpus_pitches[i:i + length]
    duration_feature = Corpus_durations[i:i + length]
    step_feature = Corpus_steps[i:i + length]
    octave_feature = Corpus_octaves[i:i + length]

    target_note = Corpus_pitches[i + length]
    target_duration = Corpus_durations[i + length]
    target_step = Corpus_steps[i + length]
    target_octave = Corpus_octaves[i + length]

    note_features.append([mapping_notes[j] for j in note_feature])
    duration_features.append([mapping_durations[k] for k in duration_feature])
    step_features.append([mapping_steps[q] for q in step_feature])
    octave_features.append([mapping_octaves[p] for p in octave_feature])

    note_targets.append(mapping_notes[target_note])
    duration_targets.append(mapping_durations[target_duration])
    step_targets.append(mapping_steps[target_step])
    octave_targets.append(mapping_octaves[target_octave])
```



B.4 Geração de uma melodia a partir de notas aleatórias

```
def Melody_Generator(Note_Count):
    seed_pitches = X_note_test[np.random.randint(0, len(X_note_test) - 1)]
    seed_durations = X_duration_test[np.random.randint(0, len(X_duration_test) - 1)]
    seed_steps = X_step_test[np.random.randint(0, len(X_step_test) - 1)]
    seed_octaves = X_octave_test[np.random.randint(0, len(X_octave_test) - 1)]

    Music = []
    Pitches_Generated = []
    Durations_Generated = []
    Steps_Generated = []
    Octaves_Generated = []

    for i in range(Note_Count):
        seed_pitches_reshaped = seed_pitches.reshape(1, length, 1)
        seed_durations_reshaped = seed_durations.reshape(1, length, 1)
        seed_steps_reshaped = seed_steps.reshape(1, length, 1)
        seed_octaves_reshaped = seed_octaves.reshape(1, length, 1)

        pred_pitches, pred_durations, pred_steps, pred_octaves =
            model.predict([seed_pitches_reshaped, seed_durations_reshaped,
                           seed_steps_reshaped, seed_octaves_reshaped], verbose=0)

        pred_pitches = np.log(pred_pitches) / 1.0
        pred_durations = np.log(pred_durations) / 1.0
        pred_steps = np.log(pred_steps) / 1.0
        pred_octaves = np.log(pred_octaves) / 1.0

        exp_preds_pitches = np.exp(pred_pitches)
        exp_preds_durations = np.exp(pred_durations)
        exp_preds_steps = np.exp(pred_steps)
        exp_preds_octaves = np.exp(pred_octaves)

        pred_pitches = exp_preds_pitches / np.sum(exp_preds_pitches)
        pred_durations = exp_preds_durations / np.sum(exp_preds_durations)
        pred_steps = exp_preds_steps / np.sum(exp_preds_steps)
        pred_octaves = exp_preds_octaves / np.sum(exp_preds_octaves)

        index_pitch = np.argmax(pred_pitches)
        index_duration = np.argmax(pred_durations)
        index_step = np.argmax(pred_steps)
        index_octave = np.argmax(pred_octaves)

        indN_pitch = index_pitch / float(L_symb_notes)
        indN_durat = index_duration / float(L_symb_durations)
        indN_step = index_step / float(L_symb_steps)
        indN_octave = index_octave / float(L_symb_octaves)

        Pitches_Generated.append(index_pitch)
        Durations_Generated.append(index_duration)
```



```
Steps_Generated.append(index_step)
Octaves_Generated.append(index_octave)

Music_pitches = [reverse_mapping_notes[char] for char in Pitches_Generated]
Music_durations = [reverse_mapping_durations[char]
                    for char in Durations_Generated]
Music_steps = [reverse_mapping_steps[char] for char in Steps_Generated]
Music_octaves = [reverse_mapping_octaves[char] for char in Octaves_Generated]

Music = chords_n_notes(
    Music_pitches, Music_durations, Music_steps, Music_octaves
)

seed_pitches = np.insert(seed_pitches, len(seed_pitches), indN_pitch)
seed_pitches = seed_pitches[1:]

seed_durations = np.insert(seed_durations, len(seed_durations), indN_durat)
seed_durations = seed_durations[1:]

seed_steps = np.insert(seed_steps, len(seed_steps), indN_step)
seed_steps = seed_steps[1:]

seed_octaves = np.insert(seed_octaves, len(seed_octaves), indN_octave)
seed_octaves = seed_octaves[1:]

Melody_midi = stream.Stream(Music)
return Music_pitches, Music_durations, Music_steps, Music_octaves, Melody_midi
```



Referências

- [1] Chih-Fang Huang and Cheng-Yuan Huang. Emotion-based ai music generation system with cvae-gan. In *2020 IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE)*, pages 220–222, 2020.
- [2] Aozhi Liu, Jianzong Wang, Junqing Peng, Yiwen Wang, Yaqi Mei, Xiaojing Liang, Zimin Xia, and Jing Xiao. Composer4everyone: Automatic music generation with audio motif. In *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, pages 502–503, 2019.
- [3] Carlos Cancino Chacón, Maarten Grachten, Werner Goebl, and Gerhard Widmer. Computational models of expressive music performance: A comprehensive and critical review. page 25, 10 2018.